How to make a word search application?

A popular *leet code*[word search](#)question we encounter these days is to create a program that would assist us to find words from a long text. Usually, a 2D board of text is provided along with the word we need to search if it exists in the grid. Solving these *word search puzzles* helps us to pass the basic test cases.

The word that needs to be searched consists of letters that are arranged sequentially in adjacent cells which are the horizontal or vertical neighbors of the word, and not in the zig-zag alignment of course. The same cell in the *word search game* cannot be searched more than once. The 8 directions in which the words can be matched are:

- Horizontally left
- Horizontally right

- Vertically up
- Vertically down, and
- The 4 diagonal directions.

The word search application also needs to return whether the word that has been searched exists in the given board or not.

For example:

board [][] = [['A', 'N,', 'J', 'K'],

['P', 'L', 'H', 'D'],

['E', 'Z', 'S', 'V']]

If,

Given word = "APE", return **true.**

Given word = "JHDV", return **true.**

Given word = "AHS", return **false.**

Let's look at the Backtracking Algorithm

In considering the backtracking algorithm to find the location of a word in a *puzzle word search*, we need to traverse the given board set *printable word search* the index as zero for the location of the word we find for the first time.

- Then from this, a recursive call is created from the current coordinate of the vector pair.
- We need to create a visited coordinate array so the recursive call is not repeated again for the previously extracted coordinate.
- If the current coordinate provides us with a solution, we need an increment in the length variable and check for a possible move in each of the feasible directions.
- If this provides us with a solution, we need to make the recursive call from that coordinate again.

- If none of these coordinates help us get the solution, we need to backtrack by marking the present coordinate as false and also decreasing the length variable.
- The whole matrix can be repeated for further results.

Let's analyze the time complexity

Now we shall discuss the time complexity analysis for *word search*. Beginning with a particular location or position of a point in the *word search printable*, it is possible to move toward eight different positions. If F(n) denotes the function carried out by checkExist(); it means the function is calling itself eight times according to the eight different directions. Now the question arises, how many times will it call itself? It will be the length of the word (denoted by K) that would have already been found by then. Since all the cells or positions are being visited and traversed in eight directions, the time complexity will be O(R*C), where R and C are the sides of the matrix. Talking about the space complexity, it is O(1), as no extra space is necessitated by the approach followed.

Conclusion

Leet code word search application is something in trend for a long time and will of course be on the top trending list for the next more years too as it is something very necessary in our daily lives. This article discusses word search and its applications, the approach to attain the proper functioning of the Leet code, and how all the cells in all eight directions are traversed. Finally, we also saw the time and space complexity of the approach. To gain deeper knowledge, refer to more of our **Skillslash** blogs and courses on **Data Science Course In Bangalore** and **Full Stack Developer Course In Bangalore**. And not to forget that our workshops are conducted by well-trained and experienced faculties who will be ready to help you out anytime in case of any doubts as we believe that everyone deserves a podium to gain knowledge and experiment with it in their lives to advance in our technological reach. Join us to get started and we can guarantee you with **100% placement** opportunities.